Université Gustave Eiffel

Cours: Services Web

Web Services Project

Car Rental Service

Work made by:
Mariana Vaz
Catarina Ribeiro
Arnaud Vanspauwen

Année universitaire 2020-2021

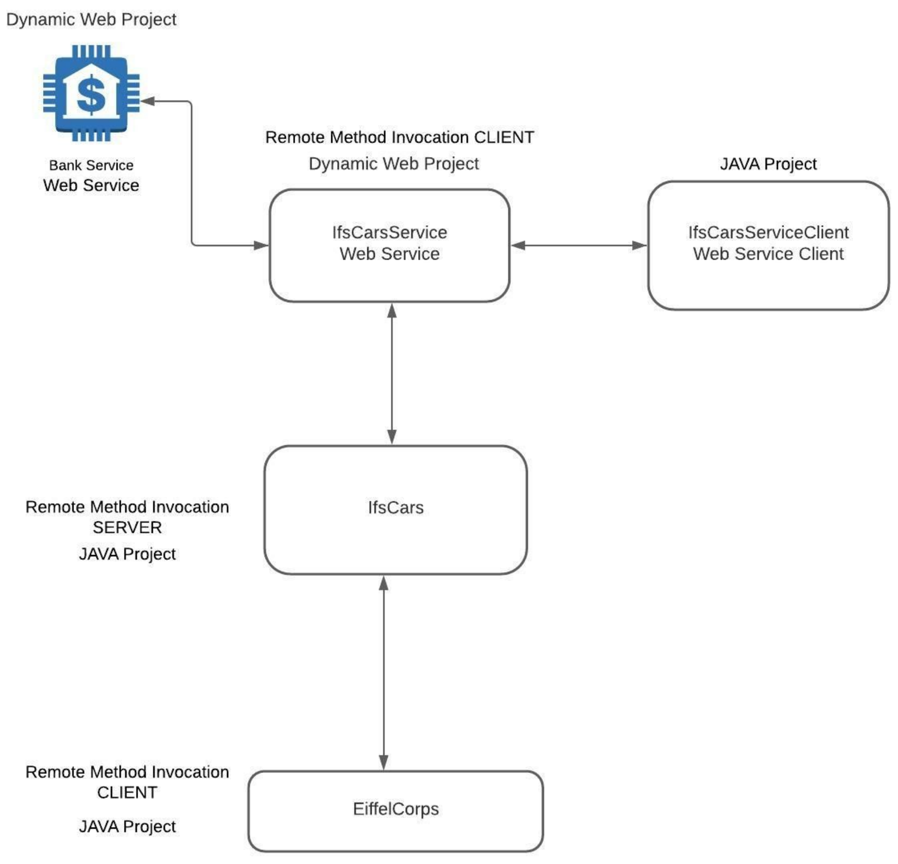
Introduction

The Car Rental Service consists in five projects that are related to each other: EiffelCorps, IfsCars, IfCarsService, IfCarsServiceClient, BankService.

Each project has its own package with its own functional classes, a common package where are the shared classes needed by all the projects and the WebServiceClient Project has the package concerning the WebService.

A RMI service is implemented to rent cars in the projects EiffelCorps and IfsCars. In the remaining classes are implemented two WebServices to sell cars.

To share all the code with each other we used git repositories.



1- General Project Structure

IfsCars

IfsCars contains four main classes that take care of the rental service based on a RMI Server:

1. **Car** with all the attributes regarding car specifications (model, brand,…)
2. **Garage** with various lists of cars and the methods implemented to rent and return cars.
3. **RentCar** that specifies what attributes define a rent.
4. **Server***

In the class Server* the following methods are used:

- createRegistry(int port) where is created and exported a registry instance on the local host that accepts requests on the specified port.
- setProperty(String key, String value) where is setted the system property indicated by the specified key
- Naming.rebind(String name, Remote obj) where the server object registers with the naming of its JVM.

Basically this application manages the renting of the cars by EiffelCorp employees or non employees. In case a person is an employee, he/she has a boolean attribute “isEmployee” that defines if he/she has a discount or not when renting a car. Also, in the renting process a person can be added to a waiting list if the car chosen it’s already being rented by someone else. When a car is returned the first person on the waiting list is notified and will rent the car. This is possible because the class implements an Observable design pattern and overrides the method notifyObserver().

EiffelCorp

This Java Project is a RMI Client and contains only one class:

1. **Client**, where the connection between the client and the server happens. It’s where we initialize the list of cars and persons and where we test all the methods that were implemented before like renting and returning a car. It’s used as well the method Naming.lookup(String name) that returns a reference, a stub, for the remote object associated with the specified name.

In this class we are also defining a codebase_path to look for the .class of the stub and a policy_path to define a security policy for the security manager.

IfsCarsService

IfsCarsService is a WebDynamicProject which implements a WebService related with the class GarageSeller, this is the main class of this project and gives to the user the possibility to see the prices and availability of each car and also the possibility to buy the car the user wants. At the same time is a WebServiceClient of the WebService implemented on the BankService and a RMI Client of the RMI Server present on the IfsCar project. The WebService is not compatible with methods that return List<> type, so the method purchaseCar has a String as an attribute to represent the basket, which is created on the client (IfCarsServiceClient) side by a StringBuilder.

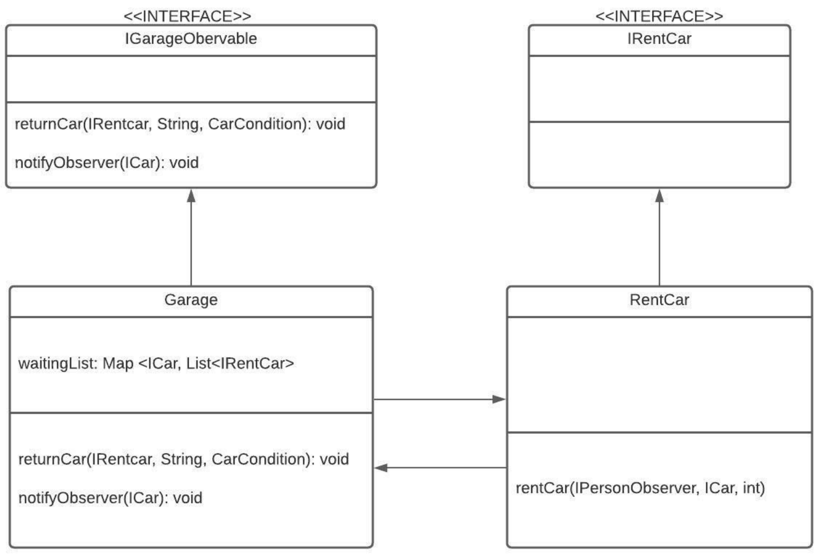
IfsCarsServiceClient

IfsCarsServiceClient is a JavaProject which implements a WebServiceClient of the WebService implemented on the IfsCarsService. The main classes in this project are the Basket and the ClientStub. The Basket is a List of ICar and the user can add all the cars he wants to buy, at the same time it is possible to delete cars from the basket and check the final price before the payment. In the ClientStub we simulate the service by purchasing a car in the choosen currency by the user. To purchase the car we access the garage created in the IfsCars RMI Server, next a new basket is created for the user and the car is purchased.

BankService

BankService is a WebDynamicProject which implements a WebService related with the class BankService, this class and the BankAccount are the main classes of this project. At the same time is a WebServiceClient of the WebService of the real time currency system. This system provides the accounts to be possible for the users to purchase the cars. Here we check if the user has enough money and in which currency he wants the price of the car.

Observer and Observable pattern


<<INTERFACE>>
IGarageObervable
returnCar(IRentcar, String, CarCondition): void
notifyObserver(ICar): void
<<INTERFACE>>
IRentCar
Garage
waitingList: Map <ICar, List<IRentCar>
returnCar(IRentcar, String, CarCondition): void
notifyObserver(ICar): void
RentCar
rentCar(IPersonObserver, ICar, int)

2 - Observer and Observable structure

In the project we were able to use some design patterns for making it easier to ourselves during the development and for future changes. We mainly used the interface pattern. We used Observer for the "first come, first served" principle (waiting list). When the person requests a car that is already on loan and he will be put in the queue. When that specific car will be returned, the first person in the waiting queue will be notified and assigned to that specific car.

Testing

For testing all the code we used the console. In the class eiffelcorp.Client we create three users, three cars and two of them are rented, it means that they will be available for sale. In the class ifscarsserviceclient.ClientStub we add the pretend car to the basket and the car is purchased in a different currency.

Conclusion

Overall we think that this was a very interesting project because it was the first time we worked with RMI's. Although, we encountered some difficulties throughout the making of the project such as:

- How to understand the structure and the division of the different projects;
- Difficulties in the github repository because we were sharing five different projects between us and some of us had different libraries and eclipse configurations;
- Also we had some difficulties importing libraries to make some classes compatible with our Eclipse;
- As mentioned before in the IfsCarsService class, to exchange objects between server and client there were some problems related to the JAX-RPC limitations.